**概念**

### **ClaimsPrincipal**

### 一个ClaimsPrincipal代表了一个用户，一个ClaimsPrincipal拥有多个ClaimsIdentity

### **ClaimsIdentity**

ClaimsIdentity代表了用户的身份证明，如身份证，一个ClaimsIdentity拥有多个Claim

### **Claim**

一个Claim代表一个信息。如身份证号，性别等

## **AuthenticationHandler认证处理器**

认证处理器提供了验证用户身份的功能，Scheme是认证处理器的标识

public interface IAuthenticationHandler

{

    // 认证，根据 HttpContext 认证用户身份并返回认证结果

    Task<AuthenticateResult> AuthenticateAsync ();

    // 如果访问的Url需要认证但用户未认证会调用该方法，一般返回 401（未认证）

    Task ChallengeAsync (AuthenticationProperties properties);

    // 如果用户没有权限会调用该方法，一般返回 403（未授权）

    Task ForbidAsync (AuthenticationProperties properties);

    // 初始化认证处理器

    Task InitializeAsync (AuthenticationScheme scheme, HttpContext context);

}

**SignInAuthenticationHandler登录认证处理器**

SignInAuthenticationHandler继承了AuthenticationHandler并添加了两个方法SignInAsync()，SignOutAsync()

    // 登录

    public virtual Task SignInAsync (ClaimsPrincipal user, AuthenticationProperties properties);

    // 登录的具体过程，由具体类实现

    protected abstract Task HandleSignInAsync (ClaimsPrincipal user, AuthenticationProperties properties);

    // 退出登录

    public virtual Task SignOutAsync (AuthenticationProperties properties);

    // 退出登录的具体过程，有具体类实现

    protected abstract Task HandleSignOutAsync (AuthenticationProperties properties);

**RemoteAuthenticationHandler远程认证处理器**

远程认证处理器继承了AuthenticationHandler 和IAuthenticationRequestHandler

public interface IAuthenticationRequestHandler : IAuthenticationHandler

{

    // 处理远程认证回调

    // 如：

    // 在OAuth认证中，浏览器在认证服务器通过认证后重定向回我们的服务器并携带了令牌

    // 在HandleRequestAsync中，拿着这个令牌去认证服务器获取用户信息

    // 然后调用 Context.SignInAsync 在我们服务器上进行登录

    // 然后重定向浏览器会之前的页面

    // 然后返回true，表示中止线程（因为远程认证回调并不是一个用户请求）

    Task<bool> HandleRequestAsync ();

}

## **AuthenticationMiddleware认证中间件**

认证中间件执行：

1. 调用所有的RemoteAuthenticationHandler.HandleRequestAsync ()，如果有返回true的，直接返回，否则直接下一步
2. 调用默认AuthenticationHandler.AuthenticateAsync()，获取用户对象ClaimsPrincipal
3. 将用户对象设置到上下文中context.User = result.Principal

## **认证与授权的流程**

下面展示一个Cookie认证的例程：

1、请求到达了服务器。

2、认证中间件调用了默认的handler上的Authenticate方法并且将任何有用的信息装在到了HttpContext.Use对象上面。

3、请求到达了控制器的action上面。

4、如果action没有被[Authorize]装饰，就直接通过请求并显示相应页面，结束。

5、如果action被[Authorize]修饰的话，授权过滤器（auth filter）检查用户是否已经认证过了。

6、如果用户没有认证，授权过滤器（auth filter）会执行Challenge动作，重定向到合适的登陆认证页面。

7、一旦登陆成功后将用户重定向回去，授权过滤器检查用户是否被允许（authorized）访问相关页面。

8、如果用户被允许访问，就展示这个页面，否则会调用Forbid动作。